End of Result Set

Generate Collection | Print

L4: Entry 8 of 8 File: USPT Jan 3, 1995

US-PAT-NO: 5379430
DOCUMENT-IDENTIFIER: US 5379430 A

TITLE: Object-oriented system locator system

DATE-ISSUED: January 3, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Nguyen; Frank T.	Campbell	CA		

US-CL-CURRENT: 707/3; 713/2, 717/111, 717/113, 717/120

ABSTRACT:

A method and system for adding system components (documents, tools, fonts, libraries, etc.) to a computer system without running an installation program. A location framework is employed to locate system components whose properties match those specified in a search criteria. The framework receives notification from the system when system components whose properties match the search criteria are added to or removed from the system.

23 Claims, 11 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 11

L4: Entry 1 of 8 File: USPT Apr 16, 2002

US-PAT-NO: 6374308
DOCUMENT-IDENTIFIER: US 6374308 B1

TITLE: Method and apparatus for interactively connecting distributed objects to a graphic user interface

DATE-ISSUED: April 16, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Kempf; James	Mountain View	CA		
Minasandram; Malini	Los Angeles	CA		

US-CL-CURRENT: 709/316; 345/762, 717/105, 717/111

ABSTRACT:

A connection is dynamically created between a graphic user interface (GUI) and a statically typed, distributed object using the present invention. The connection is created without requiring a developer to write customized code. The surrogate object acts as an intermediary between the GUI and the distributed object. The surrogate object is created using declarative information specified at build time and information generated at run time. The GUI and statically typed, distributed objects are connected via the surrogate object

20 Claims, 6 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 6

L4: Entry 2 of 8 File: USPT Oct 3, 2000

US-PAT-NO: 6128771
DOCUMENT-IDENTIFIER: US 6128771 A

TITLE: System and method for automatically modifying database access methods to insert database object handling instructions

DATE-ISSUED: October 3, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Tock; Theron D.	Sunnyvale	CA		
Cattell; Roderic G. G.	Los Altos	CA		

US-CL-CURRENT: 717/111; 707/1, 717/116

ABSTRACT:

A system and method for automatically converting a compiled program that accesses objects stored in main memory into a program that accesses and updates persistently stored objects. An initial computer program includes original instructions for accessing and updating objects in at least a first object class. The original instructions access and update objects in a computer's main memory. The system automatically revises the initial computer program to generate a revised computer program by adding to the original instructions object loading instructions and object storing instructions. During execution of the revised computer program, the object loading instructions load a copy of one of the persistently stored objects into a corresponding object in the computer's main memory when the object is accessed for a first time. The object storing instructions copy objects in the computer's main memory that contain new or modified data into corresponding persistently stored objects upon the occurrence of predefined events, such as the completion of a transaction. The system further revises the initial computer program to generate the revised computer program by adding to the original instructions dirty object marking instructions that, during execution of the revised computer program, keep track of which objects in the computer's main memory contain new and/or updated data. The object storing instructions copy only those of the objects in the computer's main memory that contain new and/or updated data.

21 Claims, 7 Drawing figures
Exemplary Claim Number: 5
Number of Drawing Sheets: 7

L4: Entry 3 of 8 File: USPT Feb 2, 1999

US-PAT-NO: 5867709
DOCUMENT-IDENTIFIER: US 5867709 A

TITLE: Method and system for reusing customizations to a software product

DATE-ISSUED: February 2, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Klencke; David L.	Denver	CO		

US-CL-CURRENT: 717/111; 717/108, 717/140

ABSTRACT:

The system described herein provides for the reusability of customizations to a software product. Once customizations are made to a software product, using the system described herein, if the software product is revised, the customizations made to the software product can be reused with little or no effort. The system described herein also facilitates the process of customizing the software product. In addition, the system described herein allows customizations made to one portion of the software product to be used by other portions of the software product. The system described herein performs its functionality by utilizing various object-oriented techniques. Object-oriented software products typically comprise a hierarchy of classes. For each class within the software product, the system described herein provides a corresponding empty class that is customizable. As such, the system described herein provides a class hierarchy for a software product that is divided into pairs of classes. Each pair of classes contains a standard class and an empty or shell class. The standard class contains functions and data to perform the functionality of the software product and the shell class contains customizations added by a customer. The customer of the software product may add whatever customizations they desire into the shell classes, which will then augment or supersede the processing of the corresponding standard class. The two classes within each pair of classes have a hierarchical relationship in which the standard class is the parent and the shell class is the child. In accordance with the system described herein, the pairs are arranged in the hierarchy in such a manner as to allow for the use of customizations within one shell class to be used by another. This hierarchical arrangement provides that when two pairs are to be placed in a parent-child relationship in the hierarchy, the shell class of the first pair is made a parent (i.e., base class) to the standard class of the second pair.

6 Claims, 6 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 6

L4: Entry 4 of 8 File: USPT Nov 24, 1998

US-PAT-NO: 5842020
DOCUMENT-IDENTIFIER: US 5842020 A

TITLE: System, method and article of manufacture for providing dynamic user editing of object oriented components used in an object oriented applet or application

DATE-ISSUED: November 24, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Faustini; Antony Azio	Palo Alto	CA		

US-CL-CURRENT: 717/111; 345/866, 345/967, 717/108, 717/113

ABSTRACT:

Method, system and article of manufacture for dynamic editing of object oriented components used in an object oriented applet or application. An editor window is defined in predetermined class templates as a method corresponding to the editor. Then, when a component is instantiated from one of said predetermined classes, the editor is automatically opened to permit the user to make changes in the component's properties. When editing is completed, the editor window is closed, the changes are accepted and then displayed for the edited component. Components are thereafter monitored for a user re-editing request which, when detected, causes the editing cycle to be initiated.

20 Claims, 33 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 32

L4: Entry 7 of 8 File: USPT Sep 24, 1996

US-PAT-NO: 5560014
DOCUMENT-IDENTIFIER: US 5560014 A

TITLE: Dynamic object management method in object oriented language

DATE-ISSUED: September 24, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Imamura; Satoshi	Tsuchiura			JP

US-CL-CURRENT: 717/108; 717/111, 717/165

ABSTRACT:

In an object oriented language, a class object is modified by using meta operators to create a floating class object by copying from a class object, modifying the floating class object into a new class object and link the new class object with its original class object by a link describing both the history of the modification and the relationship between the new class object and the original class object, and an instance object is modified by using meta operators to create a floating instance object by modifying the existing instance object and linking the modified instance object with its original class object by a link describing the modification and the relationship between modified instance object and the original class object.

6 Claims, 6 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 4 .

L4: Entry 5 of 8 File: USPT Oct 21, 1997

US-PAT-NO: 5680619
DOCUMENT-IDENTIFIER: US 5680619 A

TITLE: Hierarchical encapsulation of instantiated objects in a multimedia authoring system

DATE-ISSUED: October 21, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gudmundson; Norman K.	San Mateo	CA		
Forsythe; R. Hamish	Palo Alto	CA		
Lee; Wayne A.	San Mateo	CA		

US-CL-CURRENT: 717/108; 707/500.1, 707/515, 717/109, 717/111

ABSTRACT:

An application development system, optimized for authoring multimedia titles, enables its users to create selectively reusable object containers merely by defining links among instantiated objects. Employing a technique known as Hierarchical Encapsulation, the system automatically isolates the external dependencies of the object containers created by its users, thereby facilitating reusability of object containers and the objects they contain in other container environments. Authors create two basic types of objects: Elements, which are the key actors within an application, and Modifiers, which modify an Element's characteristics. The object containers (Elements and Behaviors--i.e., Modifier containers) created by authors spawn hierarchies of objects, including the Structural Hierarchy of Elements within Elements, and the Behavioral Hierarchy, within an Element, of Behaviors (and other Modifiers) within Behaviors. The system utilizes an Element's dual hierarchies to make that Element an environmental frame of reference to the objects it contains. Through techniques known as Hierarchical Message Broadcasting, Hierarchical Variable Scoping and Hierarchical Relative Positioning, objects automatically receive messages sent to their object container and access data known to their object container. An Element's position is even determined relative to the position of its parent Element container. The system is highly extensible through a Component API in which Modifiers and Services that support them can be added and integrated seamlessly into the system. The system's architecture is substantially platform-independent, automatically allowing most author's titles to run on multiple platforms. In addition, the entire authoring environment can be ported relatively easily to a variety of platforms due to the isolation a platform-dependent layer within the system.

48 Claims, 72 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 46

L4: Entry 6 of 8 File: USPT Feb 11, 1997

US-PAT-NO: 5603034
DOCUMENT-IDENTIFIER: US 5603034 A

TITLE: Graphical resource editor for software customization

DATE-ISSUED: February 11, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Swanson; Sara J.	Mountain View	CA		

US-CL-CURRENT: 717/111; 345/781, 345/866, 717/113

ABSTRACT:

A graphical resource editor for selectively modifying graphical resources in software applications provides a main window graphical user interface object for interaction with the graphical resource editor. The main window contains a resource category selection object including a list of selectable resource category objects. These objects contain resource category descriptors corresponding to categories of editable resources. The resource category selection object provides a user activatable interface for selecting among the list of resource category objects an editable resource category. The graphical resource editor further includes a system responsive to user activation of the resource category selection object for generating in the main window a list of resource descriptors corresponding to selected category of editable resources. A plurality of resource value display fields are provided in the main window for displaying resource values representing the status of the selected category of editable resources. Also provided in the main window is a set of resource value selection objects providing user activatable interfaces for setting editable resource values. Customization of software applications may performed statically by saving resource edits to an application resource file, or dynamically by applying resource edits on-the-fly to an application running concurrently with the graphical resource editor.

3 Claims, 18 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 14

End of Result Set

Generate Collection | Print

L3: Entry 1 of 1 File: USPT Oct 21, 1997

DOCUMENT-IDENTIFIER: US 5680619 A
TITLE: Hierarchical encapsulation of instantiated objects in a multimedia authoring system

DATE FILED (1):
19950403

Brief Summary Text (80):
Table V. Sample Instance Data Structure

Brief Summary Text (134):
Programmers also can use a process known as "inheritance" to "specialize" or modularly extend an object's functionality by reusing some or all of that object's characteristics via its interface or class definition--i.e. , the public methods and data structures that define the template for instances of that class. For example, programmers may wish to extend the base functionality of an existing class (whether created by the programmer or provided in a reusable class library) by adding a new method to objects in that class, or replacing (i.e., overriding) or supplementing (i.e., overloading) an existing method.

Brief Summary Text (177):
Both Elements and Behaviors are "object containers"--in this embodiment, object instances that can "contain" (i.e., be linked to) other object instances. Elements can contain Modifiers as well as other Elements; and Behaviors can contain Modifiers, including other Behaviors.

Detailed Description Text (23):
To create complex objects, authors need not create abstract "classes" or object templates that serve to relate objects to one another with respect to their characteristics. Elements, Behaviors and other Modifiers created by the author are object instances. To permit Elements and Behaviors to "contain" other objects, the system links the Element and Behavior object instances to the other object instances contained within them (as is explained in greater detail below with respect to the system's implementation).

Detailed Description Text (24):
Thus, Elements and Behaviors are object containers--in this embodiment, object instances that can "contain" (i.e., be linked to) other object instances. To some extent, however, Elements do attain (at least temporarily) the characteristics they contain. Yet Elements merely provide an environmental frame of reference to their descendant Elements. Elements and Behaviors do not actually "inherit" the characteristics of the objects they contain.

Detailed Description Text (111):
The structure view 33 allows authors to move Elements and Modifiers throughout the Structural and Behavioral Hierarchies--i.e., across Projects, Subsections and Scenes, as well as within Elements and Behaviors. Structure view 33 generates and controls the operation of structure view window 330, shown in FIG. 4. The Structural Hierarchy is plainly evident in that Project "Untitled-1" 303 is at the top, with Section "untitled section" 304 immediately below and indented with respect to the Project 303 icon, Subsection "untitled subsection" 305 immediately below and indented with respect to the section 304 icon, Scene "untitled Scene" 306 immediately below and indented with respect to the Subsection 305 icon. In this instance the Scene 306 icon represents a Shared Scene, to which a Graphic Modifier

308 is attached. The subsection 305 has another child scene 307, being the actual first Scene, which is represented as equally indented as its sibling Scene 306. Under this Scene 307 is Media Element 300. Knobs 302a, 302b, 302c, 302d, 302e and 302f are used to reveal or conceal the objects beneath Elements 303, 304, 305, 306, 307 and 309, respectively, in the Structural Hierarchy. This is particularly useful for an author who wishes to concentrate on only part of the Project at a given time.

Detailed Description Text (121):
An alias is a programming tool that enables authors to maintain identical settings across multiple Modifiers. When a Modifier is "aliased" initially, a master copy is placed in the Alias Palette 80. Authors can create additional aliases that refer to that master copy merely by duplicating any aliased Modifier. The advantage of using aliases lies in the fact that they are globally updatable. Changing the settings of one "instance" of an alias alters the settings of its master copy in the Alias Palette 80 and automatically updates the settings of each of the other instances of that alias throughout the Project.

Detailed Description Text (123):
With reference to FIG. 7, Alias Palette 80, controlled by Alias Manager 8, includes a close box 854, a trash can icon 851 which allows authors to remove aliases from the palette by dragging them to the trash can, and, as shown here, multiple aliases 852a, 852b, 852c, 852d, 852e and 852f, with respective user counts 853a, 853b, 853c, 853d, 853e and 853f. The user count reveals the number of instances of such alias within the current Project. Thus, if a new alias is created, a new entry in the Alias Palette 80 is added. If a modifier is de-aliased or deleted, the corresponding user count on the Alias Palette 80 is decremented.

Detailed Description Text (137):
With further reference to FIG. 2, modifier palettes 35a and 35b are shown. As discussed below, the system's extensible architecture enables programmers to create and insert additional Modifiers into the system, possibly necessitating additional modifier palettes. Each entry in a modifier palette allows an instance of a Modifier to be created and incorporated within (i.e., linked to the instance of a particular Element (Element, Scene, etc.) through a simple click, drag and drop procedure.

Detailed Description Text (275):
With reference to FIG. 16(a), the Simple Motion Modifier 1600 initiates a simple motion path. It commences and terminates execution of such motion when the messages set through pop-up menus 1603 and 1604, respectively, are received. The types of motion available are set through pop-up menu 1605 and include: (i) "Into Scene" which moves the Element into the Scene from its current position; (ii) "Out of Scene" which moves the Element off the Scene from its current position; and (iii) "Random Bounce" which bounces the object at random angles within the boundary of its parent Element. Direction in the first two instances of motion is specified through pop-up menu 1606. The number of steps and rate to accomplish this motion path are specified through pop-up menus 1607 and 1608, respectively.

Detailed Description Text (316):
The alias manager 8 handles the aliasing of Modifiers. For Variables, such as integers, this link of data is maintained at Runtime. Thus, changes to one instance of the aliased Variable are reflected in all other such instances during Runtime Mode. In one embodiment, aliasing affects other non-Variable Modifiers only during Edit Mode. Invocation of one instance of an aliased Modifier (e.g., in response to a message) does not result in the invocation of other such instances. However, nothing in the embodiment discussed here precludes implementing aliasing such that all Modifiers are linked dynamically, and thus are invoked simultaneously during the Runtime Mode.

Detailed Description Text (328):
Thus, an Element has (i.e., in its instance data structure) pointers to: (i) its parent Element; (ii) its first child Element; (iii) its previous sibling Element; (iv) its next sibling Element; and (v) its first child Modifier.

Detailed Description Text (383):
However, the author might not want to reference by name, as for example, in the interests of reusability. Thus, the author can reference a target by "tag," a stand-in for most of those very same destinations presented in the Destination pop-up by name, and as described above as Relative Targeting in conjunction with

Table II. Unlike the bootstrapping process discussed above, this process resolves the ID codes by proceeding directly up through the Structural and Behavioral Hierarchies, accessing the nearest-neighbor sibling Elements directly from the Element's next and previous Element pointers, rather than by the "identification request broadcast" procedure. Once the proper ID code has been resolved at Runtime in the first instance, the ID code can be stuffed into f.sub.-- targetinfo, allowing the messaging to proceed in a targeted manner thereafter.

Detailed Description Text (384):
The embodiment discussed above permits targeted messaging to initial destinations within a somewhat fixed scope. It is suitable in most instances where there is a close relationship among Elements, or the number of Elements is rather small. To communicate to an Element outside this small family of Elements, in this embodiment, requires targeting the most recent common ancestor Element. The message will cascade down to that Element's progeny, including the real target Element.

Detailed Description Text (392):
Thereafter, all accumulated draw requests are performed at step 260. Each redraw command is specified by invalidating the old and new visible regions of the corresponding Element. As described above, that may be driven by posted requests from the Elements, or by Motion Modifiers, etc. There may be instances where particular Elements need to be notified that such drawing operations occurred. This decision is represented by step 261. For example, a QuickTime.TM. movie needs to know when to start its sound. Thus, such Elements would be so notified at step 262.

Detailed Description Text (399):
The core 99 in this embodiment is implemented through ordinary C++ classes. However, the modifiers 22 and Services 24 are not ordinary C++ classes. To permit extensibility of the system, they are implemented through the "MOM" object model described below. In order to handle these "new" Modifier/Service classes, "wrapper" classes 94 and 95 of the ordinary C++ type are provided to interface with them as was discussed above in the context of FIGS. 25 and 26. Through these wrapper classes, the Service manager 23 controls the Services 24 and the world manager 10 and Component manager 20 instantiates Modifiers, such as Modifier 96 illustrated in FIG. 29. The world manager 10 also instantiates Elements, such as Element 97, which are coupled to corresponding instances of a media player class 12 and an asset class 11.

Detailed Description Text (417):
where "f.sub.-- vtabs" represents the pointer to the vtabs table, i.e. the table which points to all virtual tables, from which the class' virtual table may be derived; "f.sub.-- soffsets" is a table of offsets from the root of each self record in each of the base classes; and "f.sub.-- cindx" is the Component index, which is an index maintained by the Component Manager 20 which allows the Component to access common information about such Component, such as its name. This structure is appended with the instance data.

Detailed Description Text (422):
An example of an instance data structure is presented in the case of a "gravity" Modifier, which would add "mass" to Elements, prompting them to "fall" in the presence of the gravitational "force," implemented as discussed below, through a Service. Thus, the structure for the instance data could be as described in Table V:

Detailed Description Text (442):
The MOM architecture above permits many modules inside the core 99 to be migrated easily into Components. For example, there is nothing to preclude the media player classes 12, the draw engine for the rendering manager 14, the memory manager 17, the file I/O for the object loader 16 or the database management system for the asset manager 7 from being implemented as a Service rather than being in the core 99. Thus, if one needed to scale up the system to handle an enormous number of objects, one could replace the database system of the asset manager 7 with a heavy-duty relational database system.

Detailed Description Text (512):
The state of MinFlag 2243 is toggled and the message "MinBoxActivated" is sent, along with the new state of MinFlag 2243, to the parent of MinBox 2240, in a Relative Targeting fashion. This turns out to be the window 2200 itself. Thus, this message triggers the two instances of MinMaxResponse 2207 and 2226 on window 2200

and TitleBar 2200, respectively.

Detailed Description Paragraph Table (6):

TABLE V SAMPLE INSTANCE DATA STRUCTURE Member Submember Type Description f.sub.-- enableEvent f.sub.-- type short Specifies the enabling event. f.sub.-- event long f.sub.-- event.sub.-- info long f.sub.-- disableEvent f.sub.-- type short Specifies the disabling event. f.sub.-- event long f.sub.-- event.sub.-- info long f.sub.-- mass f.sub.-- type short Specifies the Element's f.sub.-- value double "mass". f.sub.-- lastvector f.sub.-- type short Specifies the Element's f.sub.-- angle double "position". f.sub.-- magnitude double f.sub.-- pixelvelocity f.sub.-- type short Specifies the Element's f.sub.-- value.PNTX short "velocity" in pixels. f.sub.-- value.PNTY short f.sub.-- totalMoved PNTX short Specifies the total PNTY short displacement since the Element started falling. f.sub.-- startTime -- long Specifies the time the Element started falling.

Current US Cross Reference Classification (4):
717/111

End of Result Set

Generate Collection | Print

L3: Entry 1 of 1 File: USPT Oct 21, 1997

DOCUMENT-IDENTIFIER: US 5680619 A
TITLE: Hierarchical encapsulation of instantiated objects in a multimedia authoring system

US PATENT NO. (1):
5680619

Brief Summary Text (80):
Table V. Sample Instance Data Structure

Detailed Description Text (422):
An example of an instance data structure is presented in the case of a "gravity" Modifier, which would add "mass" to Elements, prompting them to "fall" in the presence of the gravitational "force," implemented as discussed below, through a Service. Thus, the structure for the instance data could be as described in Table V:

WEST

End of Result Set

Generate Collection | Print

L7: Entry 1 of 1 File: USPT Oct 21, 1997

DOCUMENT-IDENTIFIER: US 5680619-A
TITLE: Hierarchical encapsulation of instantiated objects in a multimedia authoring system

US PATENT NO. (1):
5680619

Detailed Description Text (352):
The assets of individual Scenes are stored, each in turn, with the table of assets for the given Scene as a "hidden" asset. These tables of assets written to disk by the title builder 51 would be stripped of everything unnecessary for the title. Thus, unused assets as well as the user count are discarded. The asset manager 7, in conjunction with the title builder 51, also would have scanned the Scenes and correlated the order in the asset manager 7 database with layer order number for each of the Elements in the Scene before writing these assets and the tables of assets to disk.

providing the object oriented language with a group of meta operators for modifying the class objects and the instance objects,

modifying an instance object related to a class object by using the group of meta operators to obtain a modified instance object, and

linking the modified instance object with the related class object by a link describing both a history of the modification and a relationship between the modified instance object and the related class object.

5. A dynamic object management method in an object oriented language having a hierarchy of class objects describing slots for storing data and object methods consisting of groups of algorithms for conducting operations on the data and having instance objects associated with the individual class objects, the dynamic object management method comprising the steps of:

providing the object oriented language with a group of meta operators for modifying the class objects and the instance objects,

modifying an original class object using a first modification according to the steps of

obtaining a floating class object that is a copy of a class object by using the group of meta operators,

creating a new class object by modifying the floating class object, and

linking the new class object with the original class object by a link describing both a history of the first modification and a first relationship between the new class object and the original class object,

modifying an instance object related to the class object using a second modification by using the group of meta operators to obtain a modified instance object, and

linking the modified instance object with the related class object by a link describing both a history of the second modification and a second relationship between the modified instance object and the related class object.

"database" argument p.2, 3rd para

L3: Entry 1 of 3 File: USPT Jun 3, 1997

DOCUMENT-IDENTIFIER: US 5636117 A
TITLE: Method and apparatus for monitoring the strength of a real estate market or commodity market and making lending and insurance decisions therefrom

US PATENT NO. (1):
5636117

Detailed Description Text (4):
The raw data required for the method of the present invention can normally be obtained from a real estate broker's multiple-listing service (MLS) or a realtor's association. Most geographical regions of the United States have an MLS which tracks and compiles information regarding the real estate market of the area. Information regarding the sales of property and active property listings are typically maintained as part of a database. The data required can be extracted without difficulty from the MLS database using the appropriate computer search. In particular, a personal computer with a central processing unit and random access memory (RAM) may be used to interface with the MLS database in order to extract the desired data.

Detailed Description Text (31):
In another preferred embodiment of the present invention, the market index, M.sub.i, may be modified such that changes in the "pace" of market activity are included. The pace of market activity is defined as the average marketing time (time between listing and sale) of all properties which have been sold (either closed or pending) during the period. This can be accomplished by first gathering information (obtainable from NILS database) regarding the average marketing length, A.sub.m, of a property, and then dividing M.sub.i by A.sub.m.

CLAIMS:

3. The method of claim 2, wherein all of said data sources are a multi-listing service (MLS) database.

L3: Entry 2 of 3 File: USPT May 13, 1997

DOCUMENT-IDENTIFIER: US 5630127 A
TITLE: Program storage device and computer program product for managing an event driven management information system with rule-based application structure stored in a relational database

US PATENT NO. (1):
5630127

Abstract Text (1):
A rule-based application structure utilizes rules which are stored separately from application programs. The rules are stored in a relational database as objects. A user can modify existing rules and create new rules which are then restored in the database. Because rules are stored as objects in the database, they are easy to locate and retrieve. Because the rules are separate from the application programs, modifications to the rules are easier to accomplish.

Brief Summary Text (14):
Though the invention can be applied to many types of rule-based (or even expert) systems for any type of industry or vertical market, a preferred embodiment encompasses a Risk Analysis System for a financial institution. The Risk Analysis System embodiment of the present invention performs risks and exposure calculations (both mathematical and logical) for the institution based upon the institution's risk and exposure rules, current market information, and other data stored in a Management Information Database (database). Preferably, the system is based upon International Business Machines' CMIM information architecture.

Brief Summary Text (15):
The rules are stored in a database as objects. When the system is to perform a risk calculation based upon a specific rule, the application program retrieves that rule from the database. The rule is stored in the database in the form of a table. The application program then performs a calculation based upon the retrieved rule. The primitive values used to process the rule are stored and can be modified by a user to run "what-if" scenarios. The table structure permits nested mathematical formulas using objects (including rules) as variables. More complex mathematical formulas can be stored as programs but accessed in the same manner as an object.

Brief Summary Text (16):
Another aspect of the system allows a user to modify existing rules and create new rules. The rule is presented to the user in an easily understandable table format. Using menu driven selections, the user can modify an existing rule or create a new rule. The system then converts that rule into the storage form so the rule can be stored, or restored, in the database for use by the application program.

Brief Summary Text (17):
Because the rules are stored as objects in the database, they are easy to locate and retrieve. Access to all objects is through a standard interface. The system utilizes common access programs for primitive objects. This is unlike a typical rule-based system where the rules are deeply buried in the application program and can only be located and recognized by a skilled programmer. Further, because the rules are separate from the application program, modifications to the rules are easier to accomplish. The rules can be complemented and modified by a user that has little or no programming or software expertise.

Detailed Description Text (29):
Data and rules will be stored in the GRMS database 102. The data will be stored as a relational database, preferably utilizing the IBM database DB2 or other comparable relational database, and are part of the CMIS database 102. Rules for GRMS will be stored as objects in the database. The term "object" is an abstraction (like a variable in mathematics) that represents a value that is returned by an associated retrieval program. The object concept of the present invention allows the rule to be reused for different data applicable to many types of events.

Detailed Description Text (30):
Once the calculation is performed in GRMS, the result of the calculation is passed to a result handler. The result handler will also have rules to follow for the disposition of the result. Results can cause an update to the CMIS database 102, a long report to be created and routed to one or many people of the organization, or no further action.

Detailed Description Text (40):
The GRMS system 108 is rule-enabled, in that its internal design and implementation structure supports the rule execution concept. The storage form of these rules is a relational database of the data and logic needed by the system 100 to execute the rules.

Detailed Description Text (44):
The GRMS architecture enables an object-based reference system for financial data. GRMS is a based on the IBM Corporate Management Information Model (CMIM) as the information architecture for the institution. An information architecture is a defined general structure for the institution's data. The structure consists of entities, relationships and attributes. Entities are items of data, such as "product" or "account". Within the actual database these entities have many representations; for example "product" could have representations "personal loan", "certificate of deposit", "checking account". Relationships are expressions about how the entities are related to one another.

Detailed Description Text (47):
In accessing the database, it is the values of the attributes of the entities that is the goal. A particular entity representation is located by navigating through the entities through the paths defined by the relationships of the information architecture.

Detailed Description Text (49):
These entities are important because there is a path from one, multiple or a combination of them to each entity in a database modelled on CMIM. Thus a common accessing routine is used that uses as parameters, the entity identifier and attribute identifier desired along with the values of the seminal entities listed above. The accessing program returns the value of the attribute along with a return code signifying the result of the access processing (e.g. "successful", "entity representation does not exist", etc.). In this way, a standard accessing layer to the institution's data exists and, therefore, all the attributes of the database are available via this layer. This layer is shown in FIG. 2 in the boxes labeled "Retrieve Value" 220, 222, 232, and 234.

Detailed Description Text (53):
The currency conversion rate (currency.sub.-- conv.sub.-- rate 210), in turn, is expressed as another rule: currency spot rate (currency.sub.-- spot.sub.-- rate 228) minus ("-"226) currency exchange rate (currency.sub.-- exch.sub.-- rate 230). Those two objects are obtained by direct retrieval programs 232 and 234 from the databases. The currency spot exchange is retrieved from a market information database 236, while the currency exchange rate is obtained from the CMIM database 224. The direct retrieval programs (220, 222, 232 and 234), preferably written in a structured query language (e.g. SQL), utilize the key structure defined by CMIM to access the requested values from the data base 224.

Detailed Description Text (64):
The storage form of a rule, shown in FIG. 3, is in the form of a table 300, which is

the format in which the rule is stored in the database. This is the format which application programs obtain from the data base in the same manner other objects are obtained. The application program can then process the table to execute the rule it represents.

Detailed Description Text (69):
Once the business professional's rule is edited and accepted, the system 108 loads the database 102 based upon defined table structures.

Detailed Description Text (75):
At this point, the Business Professional would want to test the formula to see the values it produces in order to confirm that it produces the desired result. The user specifies a specific product identifier (such as FXYN10735) which could represent a Foreign Exchange Option in YEN for a specific customer by customer identifier (such as CUST003002 for John Smith). This is accomplished by the user via a graphic screen interface 500 entering the identifiers and the system 108 retrieves the values of the entities needed for accessing the database and processes the rule against that data. In this case, it could be (for example):

Detailed Description Text (80):
The processor then would check for a prior retrieval of this value in its Object Instance table (for example, 400 of FIG. 4) associated with this report. Since this is the first access, the table would be empty, and the program would complete the processing of the retrieval from the actual database. Once the value is retrieved, the last action before returning would be the creation of an entry in the GRMS Object Instance Table for the value. This entry is shown as the first row 410 of the table 400 in FIG. 4. A similar processing would be done for the "Currency.sub.-- Exch.sub.-- Rate", which would result in the second entry in the table 400 of FIG. 4. Each time a rule is executed, an object instance table containing the primitive value used by the rule is created.

Detailed Description Text (84):
An additional feature of the GRMS architecture is the placement of the GRMS processor on the Business Professional's workstation 118 along with the Object Table 300, and the programs defined in the object table 300. Since the object instance table 400 is also present, the Business Professional can change values in the Object Instance table (via GRMS screens and functions) and reprocess the report on the workstation. All object accesses will be satisfied by the Object Instance table function and therefore, the CMIM database 224 is not needed for this "What if" analysis reporting.

Detailed Description Text (88):
In the execution environment, the application program executes individual rules by locating the object table of the rule stored in the database and then processing that table.

Detailed Description Text (89):
The rules are located in the same manner as other data objects stored in the data base. The application program indexes the table based on the rule name. It traverses the object table to identify all primitive objects which are related to the rule name. The primitive objects are either data access programs or complex formulas. Primitive objects do not decompose into additional objects, rather they are related to programs which access data from a database or perform complex calculations.

Detailed Description Text (90):
An example of the object table structure and specific data related to the object exposure rule is provided in FIG. 3. Within FIG. 3, an example of the data access program to access the Option.sub.-- Duration is IMDCMI22 332; and an example of a complex formula to determine the Option.sub.-- Value is Black.sub.-- Scholes.sub.-- Pgm 312. All primitive objects which are data access programs are based on the information architecture of CMIM. CMIM defines the standard key structure for processing information from the relational database. This key structure combined with a standard query language provides a uniform and consistent way to access the database 102.

Detailed Description Text (117):
The CMIS system is the embodiment of all these common needs. It provides updates to the CMIS database, modeled after CMIM. It provides many common services including communications, external system fees, and other messaging.

Detailed Description Text (118):
CMIS Database

Detailed Description Text (119):
The CMIS database is based on the CMIM data model. The model is an identification and classification of strategic information need by a financial institution. It includes information about Products, Customers, Customer Accounts, Employees, Risk, Exposure, and other diverse types of information. It was designed so that each piece of information is represented in one place in the model. All of the CMIS applications use the data stored in this common, integrated database. The CMIS database is designed to be flexible and extendable, so additional types of data can be integrated within the framework provided.

Detailed Description Text (133):
Information Handling, Financial Information Delivery Component, Integrated Database, Global Communication, GRMS Developer and Risk Engineering.

Detailed Description Text (135):
The GRMS Event Processor is driven by several kinds of events from external and internal sources. From sources external to GRMS, it receives business and market event messages from the CMIS external system feeds component and processes them according to the product life cycle definitions. These events communicate to GRMS that new information has been updated on the CMIS databases (such as movements in product balances) that necessitates the recalculation of exposure. Within GRMS, the Event Processor receives events from other components, event calculators and processors, or from the Event Calendar. The Event Calendar is a service element that accepts requests for time-triggered events and generates these events at the appropriate time. These internal events, mainly report and query requests, and behavior rules checking requests, are processed and can cause GRMS calculations to occur and data to be queued to business professionals.

Detailed Description Text (136):
The heart of GRMS is its ability to calculate exposure and probable loss for various products in various states, based on a set of risk factors. When event messages are received (from CMIS, the GRMS Event Calendar, and business professionals), the calculators are executed, and data is optionally stored to the GRMS database.

Detailed Description Text (147):
When a query is run on the host, all of the data used in running that query is stored in an "Object Instance Table." This table is sent to the business professional at his/her workstation and contains a diverse set of information: product names, customer identifiers, exposure values, risk factors-everything that was calculated as part of the query or found in any database table.

Detailed Description Text (151):
GRMS uses the CMIS database which is a relational database established and shared by other CMIS Applications (i.e., Customer Information, Product Management and Balance Sheet and Liquidity Management). The Integrated Database Component unites the various databases that comprise the GRMS system, as illustrated in FIG. 7. This figure shows configuration with multiple primary nodes 704 and 706. The Enterprise Database 702 would be the one for the entire bank while the other primary nodes' databases would reflect data for subunits of the enterprise (e.g., a domestic unit, foreign unit or foreign regional). Also illustrated is the connection 708 between a primary node (in this case the Enterprise node 702) and a workstation where the database (primary node) and the data tables 710 (secondary workstation node) are related by the Integrated Database components in each. Finally, the figure illustrates the connection of the object instance table 712 that, as a result of a query or report, is developed on the primary node and transferred to the secondary mode for use in query and report interpretation and risk analysis.

Detailed Description Text (152):
The component facilitates the exchange of database data between the Host-based CMIS database and the workstation database accessed by the Financial Information Delivery Component. In addition, it processes the GRMS rollup and rolldown information between the CMIS databases in the Enterprise node and other primary nodes.

Detailed Description Text (153):
Global Communication provides the means, within the CMIS Global Communication facilities, for the connection of several host GRMS systems to communicate and exchange transactions, events and data (e.g., messages, report and query responses, roll-up and roll-down information to be processed by the Integrated Database component).

Detailed Description Text (163):
Datastore 804 represents a functionally oriented repository for data. It may actually become a database table, an in memory table, or other representation in the technical design.

Detailed Description Text (171):
The first step is to look up the appropriate risk type identification rules for the event based on product types and other user defined criteria. Using these rules the risk types affected are determined and a separate exposure amount is created for each risk type. The results of these calculations are updated on the exposure database.

Detailed Description Text (173):
This process evaluates exposure details stored in the exposure database. It compares these exposure details against behavior rules, triggers violation reports and performs trend analysis of historic exposure data.

Detailed Description Text (177):
This process creates an exposure history by reading the exposure datastore and moving selected exposure details to the Exp Hist Data. This process is rule based, allowing the bank to determine how long exposure data should stay on the exposure database and what historic exposure details are required.

Detailed Description Text (178):
The first step reads the exposure history rules. These rules are then executed against the exposure database to select the exposure detail records which meet the selection criteria specified in the exposure history rules. The selected records are then moved from the exposure database to the Exposure History Data, resulting in a reduced exposure database with quick access to the most current records.

Detailed Description Text (195):
Data stored in the CMIM defined CMIS database. This data is required by multiple CMIS business applications. It includes bank organizational data, customer data, product attributes, account details, performance measurements, price data.

Detailed Description Text (207):
Parameters that determine what exposure details are to be removed from the exposure database and moved to the exposure history database.

Detailed Description Text (236):
This process receives report requests from reporting rules, selects data from the requested databases and produces risk reports. These reports are created when events occur (defined by the reporting rules) which meet the report criteria.

Detailed Description Text (246):
Data stored in the CMIM defined CMIS database. This data is required by multiple CMIS business applications. It includes bank organizational data, customer data, product attributes, account details, performance measurements, price data.

Detailed Description Text (283):
Data stored in the CMIM defined MIS database. This data is required by multiple CMIS business applications. It includes bank organizational data, customer data, product

attributes, account details, performance measurements, price data.

Detailed Description Text (287):
Updated object data to be sent to the CMIS database, other CMIS installations and/or secondary nodes.

Detailed Description Text (297):
Requests describing the requirements for instances of data stored within the CMIS database.

Detailed Description Text (318):
This process receives the results of queries/reports from the primary node, presents them to users, and stores the object instance data to the local database for Risk Analysis. Other activities can also be received from the host, including messages from other users, and messages from behavior rule (e.g., limit) processing.

Detailed Description Text (381):
It is not the goals of a logical database model to identify physical tables and fields. Often, however, the physical and logical models are quite similar. In the physical data design, efficiency is a major concern. A small amount of redundancy may be acceptable to achieve significant efficiency gains.

Detailed Description Text (463):
This entity relates objects to modules. Different objects are used in different situations. Process objects are used when processing a message. Data objects are used to access data values throughout the system. Calculation objects compute values using data objects. Limit objects are used to aggregate data for limit processing. The distinction between calculation and data objects is important when defining objects, but is not important when using them. For example, a process object that needs a risk factor doesn't care whether it is a calculated value, or is stored in a database table. It simply requests that the object read the value.

Other Reference Publication (8):
Victor Kulkosky, "Wall Street Builds on Relational Foundation: Relational Database Management Systems Are Helping Wall Street Turn Dribs And Drabs of Data Into Money Making Information," Wall Street Computer Review, vol. 8, No. 3, Dec. 1990, pp. 33-36.

CLAIMS:

1. A computer program product for use with a relational database for storing rules and data and for use with a display, said computer program product comprising:

a computer usable medium having a computer readable program code means embodied in said medium for enabling a computer to perform risk and exposure calculations in accordance with rules stored as objects in the relational database, said computer readable program code means comprising:

computer readable first program code means for enabling said computer to display rules to perform risk and exposure calculations in a presentation format on the display and for allowing a user to modify existing rules in a presentation format and create new rules in a presentation format using arithmetic and logical expressions;

computer readable second program code means to enable said computer to translate rules in a presentation format to rules in a table format, and for storing said rules in a table format as objects in the relational database; and

computer readable third program code means for enabling said computer to retrieve said rules in a table format stored as objects in the relational database and to perform calculations according to said rules in a table format using said stored data, said computer readable third program code means including,

computer readable fourth program code means for enabling said computer to respond to external and internal events, and

computer readable fifth program code means for enabling said computer to retrieve said stored rules and data with predetermined keys.

3. The computer program product of claim 2, wherein said computer readable third program code means further includes means for retrieving preselected ones of said rules in said table format for the relational database and performing a calculation defined by said preselected ones of said rules in said table format.

5. The computer program product of claim 3, wherein said rules stored as tables in the relational database can be retrieved as objects.

6. A computer program product for use with a relational database for storing rules, said computer program product comprising:

a computer usable medium having a computer readable program code means embodied in said medium for enabling a computer to perform calculations in accordance with rules stored as objects in the relational database, said computer readable program code means comprising:

computer readable first program code means for causing said computer to input one of a mathematical and a logical rule which defines a calculation, said rule including variables;

computer readable second program code means for causing said computer to convert said rule into a table which also defines said calculation;

computer readable third program code means for causing said computer to store said table as an object in the relational database; and

an application program for retrieving said table from the relational database: and performing said calculations defined in said table using said stored data, said application program including,

an event processing program responsive to external and internal events, and

an interfacing program having predetermined keys for retrieving said stored rules and data.

8. The computer program product of claim 7, wherein including a program for retrieving said table from the relational database and performing said calculation defined by said table.

11. A program storage device readable by a computer system, tangibly embodying a program of instructions executable by said computer system to perform method steps for enabling said computer system to respond to external and internal events, to store rules as objects in an object based computer database, to retrieve and execute the stored rules, and retrieve stored data to calculate risk and exposure variables, said method steps comprising:

(1) inputting a risk and exposure rule having variables, said rule defines a logical or a mathematical calculation;

(2) converting said inputted rule of step (1) into a table;

(3) storing said table of step (2) as an object in said database;

(4) retrieving said table of step (3) from the database;

(5) obtaining data values corresponding to each said variable of said rule in said table; and

(6) performing said calculation defined by said rule of step (1).

15. A program storage device readable by a computer system, tangibly embodying a

program of instructions executable by said computer system to perform method steps for enabling said computer system, to store rules utilized by application programs, wherein the rules are stored as objects in a database, said method steps comprising:

(1) converting a rule in the form of a mathematical or a logical formula having variables to a table format, wherein each of said variables in said rule is decomposed into a rule previously converted into said table format, a program for retrieving a primitive data value for each said variable or a program for calculating a value for each said variable; and

(2) storing said table as an object in said database.

```
S (RELATION? (3N) INSTANCE) AND ((MARKET? OR COST? OR FEE) (3W) INSTANCE) AND LINK?

Your SELECT statement is:
   S (RELATION? (3N) INSTANCE) AND ((MARKET? OR COST? OR FEE) (3W)
INSTANCE) AND LINK? AND DATABSE AND PD<=970801

           Items   File
           -----   ----
Processing
Processing
       Examined  50 files
       Examined 100 files
       Examined 150 files
       Examined 200 files
       Examined 250 files
       Examined 300 files
       Examined 350 files

   No files have one or more items; file list includes 356 files.
   One or more terms were invalid in 189 files.

?
```

```
S (TRANSACTION (3N) INSTANCE) AND (RELATION? (3N) INSTANCE) AND (MARKET? (3W) INSTAN

Your SELECT statement is:
   S (TRANSACTION (3N) INSTANCE) AND (RELATION? (3N) INSTANCE) AND (MARKET?
(3W) INSTANCE) AND LINK? AND DATABSE AND PD<=970801

           Items   File
           -----   ----
Processing
       Examined  50 files
       Examined 100 files
       Examined 150 files
       Examined 200 files
       Examined 250 files
       Examined 300 files
       Examined 350 files

   No files have one or more items; file list includes 356 files.
   One or more terms were invalid in 189 files.

?
```

```
S (TRANSACTION (3N) INSTANCE) AND (RELATION? (3N) INSTANCE) AND ((PRODUCTION OR SERV
<=970801

Your SELECT statement is:
   S (TRANSACTION (3N) INSTANCE) AND (RELATION? (3N) INSTANCE) AND
((PRODUCTION OR SERVICE) (3W) INSTANCE) AND LINK? AND DATABSE AND
PD<=970801

            Items   File
            -----   ----
       Examined  50 files
       Examined 100 files
       Examined 150 files
       Examined 200 files
       Examined 250 files
       Examined 300 files
       Examined 350 files

   No files have one or more items; file list includes 356 files.
   One or more terms were invalid in 189 files.

?
```

Search Results -

Terms	Documents
L12 and (relation with table)	6

Database:
- US Patents Full-Text Database
- US Pre-Grant Publication Full-Text Database
- JPO Abstracts Database
- EPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Search: L14

Refine Search

Recall Text | Clear

Search History

DATE: Thursday, October 24, 2002 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; THES=ASSIGNEE; PLUR=YES; OP=OR		
L14	L12 and (relation with table)	6	L14
L13	L12 and (transaction adj instance)	0	L13
L12	L11 and (relation$ adj instance)	6	L12
L11	L10 and (cost$ or pric$ or bill$)	10	L11
L10	L9 and transaction	18	L10
L9	L8 and link$ and (relation$ adj database)	33	L9
L8	L7 and "entity instance"	96	L8
L7	(entity adj instance) and @ad<=19970801	96	L7
L6	L2 and (entity adj instance)	0	L6
L5	L4 not l3	7	L5
L4	L2 and ((object-orient$) or oo$)	8	L4
L3	L2 and (relational adj database)	1	L3
L2	L1 and @ad<=19970801	18	L2
L1	instance and (707/103,104.1.ccls. or 717/111.ccls.)	26	L1

END OF SEARCH HISTORY

Generate Collection | Print

L14: Entry 2 of 6 File: USPT Dec 2, 1997

US-PAT-NO: 5694598
DOCUMENT-IDENTIFIER: US 5694598 A

TITLE: Method for mapping data between a relational format and an object-oriented format

DATE-ISSUED: December 2, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Durand; Jacques	Louisville	CO		
Ganti; Murthy	Plainsboro	NJ		
Salinas; Ricardo	Superior	CO		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
U S WEST Technologies, Inc.	Boulder	CO			02

APPL-NO: 08/ 321714 [PALM]
DATE FILED: October 12, 1994

INT-CL: [06] G06 F 17/30, G06 F 13/00

US-CL-ISSUED: 395/614; 395/611, 395/612
US-CL-CURRENT: 707/103R; 707/100, 707/101

FIELD-OF-SEARCH: 395/600, 395/400, 395/700, 395/413, 395/614, 395/611, 395/612, 364/242.94, 364/243, 364/280, 364/282.1, 364/222.81

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISS[redacted]-DATE	PATENTEE-NAME[redacted]	US-CL
☐	4205371	May 1980	Feather	395/612
☐	5239647	August 1993	Anglin et al.	395/621
☐	5263167	November 1993	Conner, Jr. et al.	395/604
☐	5291583	March 1994	Bapat	395/500
☐	5386557	January 1995	Boykin et al.	395/601
☐	5426747	June 1995	Weinreb et al.	395/413
☐	5442779	August 1995	Barber et al.	395/604
☐	5448726	September 1995	Cramsie et al.	395/614
☐	5459860	October 1995	Burnett et al.	395/612
☐	5499371	March 1996	Henninger et al.	395/702

ART-UNIT: 237

PRIMARY-EXAMINER: Kulik; Paul V.

ASSISTANT-EXAMINER: Robinson; Greta L.

ABSTRACT:

A first method is disclosed for mapping data from a plurality of objects to a relational database. The method begins with the step of generating a transit object and its structure (TO.sub.-- schema, dataGraph and dataBlocks). The next step of the method is populating the transit object based on the data of the plurality of objects. The method continues with the step of transmitting the transit object from the client object broker (COB) to the server object broker (SOB) using a communication server. The method next includes the step of populating a data structure based on the dataBlock object. The method concludes with the step of populating the relation&l database based on the data structure. A second method is also disclosed for mapping data from a relational database to a plurality of objects.

10 Claims, 15 Drawing figures

DOCUMENT-IDENTIFIER: US 5694598 A
TITLE: Method for mapping data between a relational format and an object-oriented format

DATE FILED (1):
19941012

Abstract Text (1):
A first method is disclosed for mapping data from a plurality of objects to a relational database. The method begins with the step of generating a transit object and its structure (TO.sub.-- schema, dataGraph and dataBlocks). The next step of the method is populating the transit object based on the data of the plurality of objects. The method continues with the step of transmitting the transit object from the client object broker (COB) to the server object broker (SOB) using a communication server. The method next includes the step of populating a data structure based on the dataBlock object. The method concludes with the step of populating the relation&l database based on the data structure. A second method is also disclosed for mapping data from a relational database to a plurality of objects.

Brief Summary Text (4):
The relational database model was introduced in the early 1970's by E. F. Codd. Since then, the relational model has become the model employed by most commercial database management systems (DBMS).

Brief Summary Text (5):
Data in a relational database is represented as a collection of relations. Each relation can be thought of as a table.

Brief Summary Text (6):
Like the relational database model, object-oriented programming ("OOP") has also existed since the early 1970's. In the early 1990's, object-oriented programming gained widespread acceptance due to increased power of workstations, proliferation of graphical user-interfaces and the development of hybrid object-oriented languages such as C++.

Brief Summary Text (9):
The present proliferation of relational DBMSs coupled with the increasing popularity of the OOP paradigm has resulted in a desire to map data between data models. In particular, it is desirable to access relational databases in OOP applications, and to access object-oriented data from within a relational DBMS.

Brief Summary Text (23):
A need therefore exists for an improved method for mapping data between an object oriented format and a relational format. More particularly, a need exists for a method for mapping data between an object oriented format and a relational format which provides an application with not only a facility for making objects persistent but also a facility for populating objects with data from existing relational databases.

Brief Summary Text (25):
It is an object of the present invention to provide a method for mapping data between an object oriented format and a relational format using a transit object transmitted between an object-oriented client application and a data server managing relational databases.

Brief Summary Text (29):
In carrying out the above objects and other objects of the present invention, a first method is provided for mapping data from a plurality of objects to a relational database. The method is intended for use in a data processing system which includes a processor, a memory, a client object broker ("COB"), a communication server and a server object broker ("SOB") .

Brief Summary Text (31):
The method further includes the step of populating a data structure based on at least one dataBlock object. The method concludes with the step of populating the relational database based on the data structure.

Brief Summary Text (32):
In carrying out the above objects and other objects of the present invention, a second method is provided for mapping data from a relational database to a plurality of objects. The second method begins with the step of generating in the memory a transit object. The transit object includes at least one dataBlock object.

Brief Summary Text (33):
The method continues with the step of generating in memory a data structure. The method also includes the step of populating the data structure based on the data of the relational database. The method further includes the step of populating at least one dataBlock object of the transit object based on the data structure.

Drawing Description Text (5):
FIG. 3 is a schematic block diagram illustrating a query of a relational database;

Drawing Description Text (15):
FIG. 13 is a block diagram illustrating example classes in an application domain and corresponding tables of a relational database;

Drawing Description Text (16):
FIG. 13 is a block diagram illustrating example classes in an application domain and corresponding tables of a relational database;

Drawing Description Text (18):
FIG. 15 is a block diagram illustrating the structure of an example relational database.

Detailed Description Text (10):
Security: In an object client-server architecture, security is a major concern. The application access to the database has to be restricted. For this reason, as well as for the sake of independence from the storage technology, no direct SQL capability should be made available to the application. This restriction can also be extended to read only transactions since there is a need to control the cost of databases transactions at a server level. In other words, the Object Server has a control over both content and processing of the database access.

Detailed Description Text (11):
Transaction granularity: various levels of granularity in database interaction should be accommodated (e.g. single attribute update as well as a transaction on an entire aggregation/collection of objects), and not restricted to object units. This is especially important when it comes to optimize the data flow as well as the number of transactions that is handled by a data served (and therefore its performance). Such scalability of requests for manipulating parts of objects as well as aggregations/collections of objects requires an interface more flexible than an import/export module.

Detailed Description Text (16):
Referring now to FIG. 3, there is illustrated a query of relational database 218. On the SOB 216 side, get.sub.-- SobTO, a stored procedure, is invoked by a query processor and returns the query result into an array of tuples 310.

Detailed Description Text (44):
A TO-entity contains TO-attributes of different types. A TO-entity has instances, each of which is represented as a datalist.

Detailed Description Text (48):

TO-entity Instances

Detailed Description Text (50):
The array-based implementation assigns a one-dimensional array to each TO-attribute. The array holds its instances for all TO-entity instances. The grouping of the different TO-attributes of a TO-entity can in turn be done by chaining the one-dimension arrays into a list or into a bi-dimensional array.

Detailed Description Text (52):
The list-based implementation assigns a list to each TO-entity instance (i.e. to each datalist). Therefore, it actually implements the datalist. Each element of the list, however, is actually a pointer to the value of the element. Thus, such a list can be heterogeneous having elements of different sizes.

Detailed Description Text (54):
TO-relationship Instances

Detailed Description Text (55):
The implementation of a TO-relationship requires some means to associate a TO-entity instance to zero, one or several other TO-entity instances. A TO-relationship is implicitly considered as an oriented, binary, many-to-many relationship. Given a TO-relationship r.sub.12 from a TO-entity e.sub.1 to a TO-entity e.sub.2, we call "r.sub.12 -reference" the link from an instance of e.sub.1 to an instance of e.sub.2.

Detailed Description Text (147):
In the preferred embodiment, the set of stored procedures interface on the database side. This means that there are stored procedures that match a particular application object model. While this is a violation of the independence data-store/application object-model, this is the price to pay for efficient database access.

Detailed Description Text (177):
Referring now to FIG. 13, there is illustrated the example classes CO and OI of the application domain and the corresponding tables customer.sub.-- order and order.sub.-- item of the example relational database.

Detailed Description Text (180):
Consider the database illustrated in FIG. 15. A customer order is stored in the relational database, in the following form: a CO tuple called CO.sub.1, three CP tuples called (CP.sub.1a,CP.sub.1b,CP.sub.1c), two OI tuples called (OI.sub.1a,OI.sub.1b), each of them related to two AI tuples: (AI.sub.1aa,AI.sub.1ab) for OI.sub.1a, and (AI.sub.1ba,AI.sub.1bb) for OI.sub.1b. This relational schema corresponds to a TO-schema with tables CO, CP, OI, AI.

CLAIMS:

1. In a data processing system including a processor, a memory, a client object broker ("COB") responsible for object data stored in an object-oriented application object, a communication server and a server object broker ("SOB") responsible for relational data stored in a relational database, a method for mapping data from a plurality of objects to the relational database, the method comprising:

generating in the memory a transit object corresponding to a generic intermediate form of data independent of the object data and the relational data, the transit object including at least one dataBlock object;

populating the at least one dataBlock object of the transit object based on the data of the plurality of objects stored in the COB;

transmitting the transit object from the COB to the SOB using the communication server;

populating a data structure based on the at least one dataBlock object; and

populating the relational database based on the data structure.

6. In a data processing system including a processor, a memory, a client object broker ("COB") responsible for object data stored in an object-oriented application

object, a communicat[redacted] server and a server object br[redacted]r ("SOB") responsible for relational data stored in a relational database, a method for mapping data from the relational database to a plurality of objects, the method comprising:

generating in the memory a transit object corresponding to a generic intermediate form of data independent the object data and the relational data, the transit object including at least one dataBlock object;

generating in memory a data structure;

populating the data structure based on the data of the relational database stored in the SOB;

populating the at least one dataBlock object of the transit object based on the data structure;

transmitting the transit object from the COB to the SOB using the communication server; and

populating the plurality of objects based on the at least one dataBlock object.

End of Result Set

Generate Collection | Print

L3: Entry 1 of 1 File: USPT Oct 21, 1997

US-PAT-NO: 5680619
DOCUMENT-IDENTIFIER: US 5680619 A

TITLE: Hierarchical encapsulation of instantiated objects in a multimedia authoring system

DATE-ISSUED: October 21, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gudmundson; Norman K.	San Mateo	CA		
Forsythe; R. Hamish	Palo Alto	CA		
Lee; Wayne A.	San Mateo	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
mFactory, Inc.	Burlingame	CA			02

APPL-NO: 08/ 415848 [PALM]
DATE FILED: April 3, 1995

INT-CL: [06] G06 F 9/40

US-CL-ISSUED: 395/701
US-CL-CURRENT: 717/108; 707/500.1, 707/515, 717/109, 717/111

FIELD-OF-SEARCH: 395/650, 395/700, 395/701

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5287447	February 1994	Miller et al.	395/157
☐	5423041	June 1995	Burke et al.	395/700
☐	5493680	February 1996	Danforth	395/700
☐	5499333	March 1996	Doudnikoff et al.	395/600
☐	5499371	March 1996	Henninger et al.	395/700

OTHER PUBLICATIONS

Booch, "Elements of the Object Model", Object Oriented Analysis and Design with Applications, 2nd Ed. 1994, pp. 40-77.

ART-UNIT: 236

PRIMARY-EXAMINER: Kriess; Kevin A.

ASSISTANT-EXAMINER: Chavis; John I.

ABSTRACT:

An application development system, optimized for authoring multimedia titles, enables its users to create selectively reusable object containers merely by defining links among instantiated objects. Employing a technique known as Hierarchical Encapsulation, the system automatically isolates the external dependencies of the object containers created by its users, thereby facilitating reusability of object containers and the objects they contain in other container environments. Authors create two basic types of objects: Elements, which are the key actors within an application, and Modifiers, which modify an Element's characteristics. The object containers (Elements and Behaviors--i.e., Modifier containers) created by authors spawn hierarchies of objects, including the Structural Hierarchy of Elements within Elements, and the Behavioral Hierarchy, within an Element, of Behaviors (and other Modifiers) within Behaviors. The system utilizes an Element's dual hierarchies to make that Element an environmental frame of reference to the objects it contains. Through techniques known as Hierarchical Message Broadcasting, Hierarchical Variable Scoping and Hierarchical Relative Positioning, objects automatically receive messages sent to their object container and access data known to their object container. An Element's position is even determined relative to the position of its parent Element container. The system is highly extensible through a Component API in which Modifiers and Services that support them can be added and integrated seamlessly into the system. The system's architecture is substantially platform-independent, automatically allowing most author's titles to run on multiple platforms. In addition, the entire authoring environment can be ported relatively easily to a variety of platforms due to the isolation a platform-dependent layer within the system.

48 Claims, 72 Drawing figures